
DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU


HARVARD LIBRARY
Office for Scholarly Communication

Receipt-Free Secure Elections

Citation

Ellard, Daniel and David Alpert. 2003. Receipt-Free Secure Elections. Harvard Computer Science Group Technical Report TR-13-03.

Permanent link

http://nrs.harvard.edu/urn-3:HUL.InstRepos:25619465

Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

Receipt-Free Secure Elections

Daniel Ellard
and
David Alpert

TR-13-03

Computer Science Group
Harvard University
Cambridge, Massachusetts

Receipt-Free Secure Elections

Daniel Ellard and David Alpert

Abstract

A fundamental requirement for secure and uncoercible elections is the receipt-free property; if voters cannot prove how they voted to anyone other than themselves, then they cannot be coerced by a second party to cast a particular vote.

We begin by describing a protocol for receipt-free elections that was developed by Benaloh & Tuinstra, and then show how this protocol, although correct, is impractical to implement. We then show how to modify this protocol to make it practical to implement. Our protocol requires the addition of an "uncoercible" third party who witnesses the execution of the protocol to make sure that it is executed correctly, but who does not learn the value of the vote.

1 Introduction

Secure elections have been the subject of much research, but there still remain unsolved problems, primarily in the areas of practicality and implementation. Our project focusses on the issue of receipt-free and uncoercible elections.

For example, the protocol developed by Fujioka et al [4], which is heralded as a practical algorithm, is actually very difficult to implement in a convenient manner, as Ben Adida et al discovered during their implementation. Furthermore, this protocol makes no provision for preventing the coercion of voters, and we could not see how it could be patched to add this desirable property.

The algorithm developed by Niemi & Renvall [8] uses a very different approach, employing a multi-party protocol, secret sharing, and multi-party computation to achieve non-coercibility. It bases its security on the infeasibility of corrupting all of the parties on the system– but in order for this to be effective, there must be many parties in the system! They admit in their conclusion that their protocol, although promising and theoretically interesting, is impractical.

Another approach to non-coercible votes is to use the techniques developed to implement electronic cash– in essence, each voter votes by "purchasing" the kind of vote they desire. However, these approaches are complex and generally do not seem to be well-suited to voting protocols– the desired properties of electronic cash are somewhat different than the desired properties of electronic votes.

Other investigators [10] have sacrificed uncoercibility in their attempt to achieve practicality, but we do not accept this tradeoff because the resulting protocols offer no important benefits over other coercible voting protocols. Cramer et al [3] give a protocol that shares many ideas with Niemi & Renvall and Benaloh & Tuinstra, but does not achieve non-coercibility.

We believe that the type of protocol created by Benaloh & Tuinstra holds the most promise. However, the original statement of the protocol is impractical, and so we have modified it to make it prac-

tical.

2 The Basic Protocol

Our basic protocol is based on the protocol developed by Benaloh & Tuinstra [2]. We selected this protocol because of its relative simplicity and elegance. Although this protocol has some flaws that make it impractical to implement, we believe that the principles that the protocol is based on hold the promise of a practical and uncoercible voting protocol.

This section describes the basic protocol developed by Benaloh & Tuinstra, along with commentary about the implementation difficulties presented by each part of the protocol. Note that portions of this section are excerpts of their paper [2], which are reproduced here to introduce the notation that we will use throughout the rest of the paper.

2.1 Terms and Definitions

- A *voter* is a party that should be permitted to vote in the current election.

  The Benaloh & Tuinstra protocol assumes that the identity of the voter has already been authenticated before the protocol begins, and each message in the protocol from the voter to any other party can be authenticated as coming from the voter (via digital signatures, or any other method). Because of the nature of the protocol, an eavesdropper who is able to overhear every message that the voter sends will still not be able to determine how the voter voted. The voter cannot deny anything that they say, but nevertheless can deny who they voted for.

  As we modify the protocol, we must either make sure that this property still holds, or that the voter is protected from coercion in some other manner.

- The voting *authority* is a trusted party that is responsible coordinating the voting and tallying the votes of each of the voters.

  The trust granted to the voting authority is not complete– we do not necessarily trust that the authority will follow the protocol, but in most cases if the authority attempts to cheat, this will be detected, with very high probability, by all observers (including the voter) and the voting protocol can be abandoned. (If the authority is known to be trustworthy, then the protocol can be extremely simplified.)

  However, we do trust the authority in one very important aspect– we trust that the authority will never reveal anything of the information contained in the secret message that it sends to the voter, or partial tallys. If the authority itself can be coerced into revealing even a single bit that would not otherwise be revealed by the protocol, then the voter can be coerced.

  Benaloh & Tuinstra assume that the authority will not reveal any information not otherwise revealed by the protocol (such as the contents of any private messages or their private decryption key), and show that the protocol is secure if this assumption is met. They also explore the issue of how to distribute the responsibility of the authority among several autonomous parties in such a way that even if a subset of fewer than half of the authorities reveals all their information, the vote is still secure.

  - Let $c = E(m)$ denote the probabilistic public key encryption function using the authority's private key (which is known only to the authority).

It is essential that the encryption function be probabilistic (and therefore based on a secure random number generator– a difficult assumption in and of itself). Since nearly all of the encryption is done on one-bit votes (0 for no, 1 for yes), it would be a grave error to use a deterministic encryption function– the known-text attack on E would require only one encryptions!

However, the reliance on probabilistic protocols and the secrecy of the authority combine to form a weakness in the protocol– the authority can use the probabilistic encryption as a covert channel. The authority can repeatedly encrypt the same value until the result contains a desired bit pattern that communicates some information about the encrypted message. If done carefully, this kind of covert channel can be extremely difficult to detect.

– Let $D(c)$ denote the decryption function that inverts $E(m)$, so that $D(E(m)) \equiv m$. The decryption function is only known to the authority.

The Benaloh & Tuinstra protocol has a weakness if the authority can be tricked into decrypting arbitrary ciphertexts that represent votes. However, we note that all of the ciphertexts that are ever decrypted by the authority as part of this protocol must also have been encrypted by the authority. We must make sure that any other messages decrypted by the authority (for example, during the implied by unstated authentication protocol) either be encrypted using a different key, or somehow be distinguishable from encrypted votes, so that this attack can not be used to defeat the security of this protocol.

- Let N denote a security parameter, such that any attempt by the authority to cheat will be detected with probability $P \geq 1 - (\frac{1}{2})^N$.

  The complexity of the protocol, in terms of computation and messages, is $O(N)$, so there's no need to skimp on N. We can make the probability of the authority successfully cheating vanishingly small without much additional cost.

- The *voting booth* is a construction that the voter can enter and leave. When inside the voting booth, the voter can observe the outside world and receive messages. However, the voter cannot be observed from outside the booth; nothing that happens inside the booth can be detected by anyone outside the booth until the voter emerges from the booth.

  Events that occur inside the booth cannot be recorded in a time-stamped manner by the voter. The voter could be coerced by taking a tamperproof recording device into the booth and use it to record everything they observe in a time-stamped manner. Therefore, the voter is prohibited from taking any recording device into the booth.

  The basic protocol is entirely reliant on these properties of the booth. If the voter can smuggle a message out if the booth, or record a true transcript of what they observe when inside the booth, then they can be coerced. This is unquestionably a weakness of the protocol, because it is a very difficult construction to achieve.

- The *beacon* is a trusted public source of random bits.

2.2 The Election Protocol

2.2.1 Casting a Vote

1. **Authority**:

   (a) Create $N + 1$ (0, 1) pairs.

   (b) Encrypt each (0, 1) pair with E, creating $(x_0, y_0) \cdots (x_i, y_i) \cdots (x_N, y_N)$ (where each x_i is the encryption of a 0 vote, and each y_i is the encryption of a 1 vote).

   (c) Sort each of the (x_i, y_i) pairs, creating $(\alpha_0, \beta_0) \cdots (\alpha_i, \beta_i) \cdots (\alpha_N, \beta_N)$, such that $\alpha_i = \min(x_i, y_i)$, and $\beta_i = \max(x_i, y_i)$.

   At this point, only the authority knows whether each (α_i, β_i) is the encryption of (0, 1) or (1, 0).

   (d) The authority publishes the ordered list $(\alpha_0, \beta_0) \cdots (\alpha_i, \beta_i) \cdots (\alpha_N, \beta_N)$. Any interested observer can obtain a copy, and the voter *must* obtain a copy.

2. **Voter**: The vote enters the voting booth.

3. **Authority**: For each $i \in 0 \cdots N$, send $c_i = D(\alpha_i)$ to the voter. This is done via a secure channel; only the voter will be able to learn the value of each c_i directly from the authority.

   At this point, the voter has an alleged decryption of each (α_i, β_i). However, they have no proof that this decryption is correct; the authority simply asserts that this is the correct decryption but does not provide a complete proof (or certificate) that shows that this is the correct decryption.

4. **Beacon**: The beacon generates N bits $b_1 \cdots b_N \in_R \{0, 1\}$. The bits are witnessed by all interested observers, including the voter and the authority.

   Note that the beacon only generates N bits, and as a notational convenience the subscripts of these bits begin at 1 instead of 0.

   As an extension to the original protocol, we suggest that if the beacon generates b_i string that is heavily biased towards one value (or in the worst case, consists entirely of one value), the protocol is restarted. Otherwise, the probability that the next step correctly catches a devious authority is reduced.

5. **Authority**: for each b_i generated by the beacon,

   - If $b_i = 0$, then publicly *reveal* $D(\alpha_i)$ and $D(\beta_i)$, including a complete proof that these decryptions are correct.

     To the general public, this decryption shows that each decrypted pair of votes were properly constructed; each contained a single 0 and single 1 vote.

     To the voter, this decryption also proves that the authority was behaving correctly in step 2.

   - If $b_i = 1$, then publicly *connect* (α_i, β_i) to (α_0, β_0) by providing either:
     - Proof that $\alpha_i = \alpha_0$ and $\beta_i = \beta_0$.
     - Proof that $\alpha_i = \beta_0$ and $\beta_i = \alpha_0$.

     This is done via a zero knowledge proof; it is essential that these equalities are proven without revealing the decryption of α_0 or β_0. One method of doing this requires an encryption scheme that has properties described in section 2.2.4.

To the general public, this connection shows that that each connection is possible– either each (α_i, β_i) is a fraudulent pair containing the same pair as (α_0, β_0), or else each correctly contains a (0, 1) pair (in some unknown order), as does (α_0, β_0).

The probability that the authority can create bogus (1, 1) or (0, 0) votes without being detected is tiny, since the authority must reveal approximately half the votes. Unless the authority guesses every b_i correctly (or has some way to compel the beacon to act a particular way), it is very unlikely that the authority will correctly choose the single partition of the (α, β) pairs that will allow him to successfully cheat.

6. **Voter**: The voter casts their vote by publicly announcing their choice of either α_0 or β_0 (based on which of these votes they believe contains the encrypted form of the vote they desire to cast).

2.2.2 Computing the Final Tally

Once all of the votes have been publicly announced, the authority computes a final tally. This is accomplished by combining the votes directly in their encrypted form (without ever decrypting them), and then presenting a decryption of the final result, along with a proof that the decryption is correct. Every witness can check the computation to make sure that each vote was correctly included and that the final result includes all correct votes (and no additional votes).

It is important that the authority not provide any tallys other than the final, because otherwise careful observers might be able to determine the value of individual votes based on changes in the tally. Since any observer can compute a pair of encrypted tallys on their own that include and omit a particular voter, they will be able to prove how a particular voter voted if they can ever decrypt their tallys. Therefore, it is absolutely essential that the authority never decrypt anything that it didn't encrypt (as mentioned in section 2.1).

2.2.3 Uncoercibility

In this protocol, the voter cannot be coerced because the voter cannot provide any irrefutable evidence how that they voted. Their choice of α_0 or β_0 is based entirely on their belief in the correctness of the values of c_i that are given to them by the authority. However, as long as the authority does not share the c_i list with anyone else, then as soon as the voter knows b_i, he or she can create an alternate $\hat{c_i}$ list where $\hat{c_i} = b_i \oplus c_i$. In this construction, $\hat{c_i}$ will be the same as c_i for all of the votes that are publicly opened, but the opposite for all of the votes that are connected to (α_0, β_0). This allows the voter to present evidence that their vote was the opposite of their actual vote, and this evidence cannot be refuted by anyone except the authority.

However, this irrefutability property only exists because of the voting booth– if the voter can tell an outside party what c_i is before anyone knows b_i, then they will not later be able to convincingly lie about c_i.

Interestingly, this non-coercion property extends somewhat to the authority– as long as the voter cannot tell anyone c_i before knowing b_i, the authority can *also* forge a c_i list that reverses the voter's vote, and can only be successfully challenged by the voter. However, the authority can open any vote, however, and therefore can be coerced.

2.2.4 Appropriate Public Key Encryption Schemes

Not every encryption scheme provides a method to generate zero-knowledge proofs of equality (as needed in the voting protocol) or a way to prove the sum of the votes without revealing the individual votes. Benaloh & Tuinstra provide one, and cite others.

The general property that these encryption schemes must have is a homomorphism for addition and subtraction between the encrypted and decrypted values.

In particular, if $c_1 = E(m_1)$ and $c_2 = E(M_2)$, then there must exist public, polynomial-time algorithms to compute $\otimes$ and $\oslash$, which are defined to be:

$$\begin{aligned} c_1 \otimes c_2 &= E(m_1 + m_2) \\ c_1 \oslash c_2 &= E(m_1 - m_2) \end{aligned}$$

(Note– we follow Benaloh's choice of notation here, although we found it confusing.)

Note that there is one unlikely, but potential, weakness here: to show that two ciphers c_1 and c_2 represent the same value, we show that $D(c_1 \oslash c_2) = 0$. Thus, in the process we have created yet another public encryption of zero, which might (with very small probability) collide with the value of one of the other α or β values, and therefore completely reveal a vote.

3 Eliminating the Voting Booth

The voting booth is a huge obstacle to implementing this protocol; even assuming that such a device can be built, it is certainly impossible to implement it as a program running on an ordinary computer. Therefore, at best this protocol is impractical because it is inconvenient– like current paper-ballot voting protocols, it requires the voters to travel to a special site to vote, instead of voting from their homes or offices. Therefore, it is imperative that we replace the necessity of a voting booth with something feasible.

Our approach to eliminating the need of the voting booth requires a second trusted party which is used to coordinate the communication between the voter and the authority. We call this new party the *listener* because its role is primarily to listen to the exchange of messages and make sure that the protocol is obeyed.

The tallying procedure is identical to the original algorithm. A description of the modified vote-casting protocol follows:

1. Step 1 proceeds as before; the authority creates the (α, β) pairs and publishes them.

2. The authority uses a commitment scheme to commit the values of c_i to the listener via a private channel. The authority does not reveal c_i to the listener, nor does the authority communicate at all with the voter.

3. The beacon generates N numbers $b_i \in_R \{0 \cdots 3\}$ (via flipping its coin twice per b_i).

   Note that we are introducing much more randomness into this algorithm. The extra bits will be used to "blind" the listener and prevent the listener from ever learning the actual values of c_i.

4. The authority sends c_i via a private channel to the voter.

5. The voter uses the same bit-commitment scheme to commit their copy of c_i to the lis-

tener over a private channel. The listener confirms that the commitments it has received from the authority and from the voter are identical. If they are not identical, then the protocol is aborted.

We assume the use of a deterministic commitment scheme, so that if two bit strings B_1 and B_2 are committed by an identical commitment, then $B_1 \equiv B_2$. For example, we can use a secure hash function to hash the bit string c_i, and use the resulting hash value as a commitment to those bits. (We can also use a nondeterministic schemes that ensure that $B_1 = B_2$ with very high probability.)

6. The authority reveals information about each (α_i, β_i) pair. If b_i is 0 or 1, then the same protocol as described earlier applies. However, if b_i is 2 or 3, then the authority reveals no information whatsoever about (α_i, β_i).

   This overhead is necessary to prevent the listener from directly inferring the value of c_i. Based on the information available in the original protocol, the listener can eliminate all possible c_i strings except two– the real c_i, and the $\hat{c_i}$ that the voter would forge in order to pretend that they had cast the opposite vote. With only two possibilities, the listener can easily construct both, hash each, and see which matches the commitment he received in the second step.

   In this protocol, however, approximately half of the c_i bits are still completely unknown to the listener. Therefore, the listener has many possibilities to consider. By making N large enough (and making sure that our hash function is computationally secure), we can easily make it intractable for the listener to discover c_i via brute-force search, even when the search is guided by the information revealed by the authority.

7. The voter casts their vote, as before.

3.1 Uncoercibility

As before, the voter cannot reveal anything about c_i to a third party until after the beacon has generated b_i, and therefore the voter can forge a $\hat{c_i}$ that is consistent with c_i in all verifiable ways, but reverses their vote. Only the authority or the listener can claim that $\hat{c_i}$ is a forgery (and only the authority can *prove* the forgery).

3.2 Trusting the Listener

The listener is able to determine whether or not the authority and voter correctly share the c_i string, but cannot determine what the c_i string is. Thus, we do not need to worry that the listener will directly reveal the value of c_0.

However, the voter and the listener *can* collaborate to produce a receipt for the voter's vote– if the listener reveals the commitment of c_i before the beacon has generated b_i, then the commitment of c_i pins down the voter. The voter cannot successfully forge a list of c_i values that is consistent with the public information, consistent with the commitment, and reverses their vote.

Therefore, the level of trust we must grant the listener is somewhat below the authority. The authority can reveal everything about a voter, including how they voted, without the cooperation of the voter. The listener, on the other other hand, requires the collaboration of the voter in order to reveal anything useful. We must trust that the listener will not reveal the commitment of c_i. There's nothing else that the listener *knows*, however, so there's nothing else that it can reveal.

4 Open Issues

The voter can still be coerced in our protocol by being forced to show the decryption of messages sent to the voter from the authority via their private channel. If an eavesdropper can capture a copy of these messages, they can later compel the voter to decrypt them and prove that the decryption is correct. If this happens, the voter cannot deny the contents of the message and therefore cannot conceal their vote.

One solution to this problem is to require that all voting be done on a secure, trusted network– but this makes the protocol inconvenient and impractical, since voters must vote from special sites instead of voting from whatever computer they desire. Although we have eliminated the need for the unrealistically secure voting booth, the need for physical network security remains.

We see two possible solutions to this problem:

- Use a decryption method for each c_i that allows each bit to be decrypted in either manner. The decryption step has two keys– a true key, which the voter would ordinarily use, and a "duress" key, which the voter could use to decrypt the same bit to reveal the opposite value of the true value.

  Once the voter has revealed the decryption of one c_i, and proven the correctness of this decryption (possibly by revealing one of the keys), the dual of this key can never be used. Therefore, each c_i must be encrypted with a different public key.

  This method requires the generation of a huge number of keys, but is otherwise practical, assuming that a feasible probabilistic encryption algorithm with a duress key can be constructed. We conjecture that it can be, although we have not seen one.

- Use some other form of deniable communication, such as the deniable authentication protocol being developed by Michael Rabin.

5 Conclusion

We have presented a modification of the Benaloh & Tuinstra protocol that increases the practicality of the protocol to the point where secure implementations are possible.

References

[1] Baraani-Dastjerdi, A. Pieprzyk, and J. Safavi-Naini. A secure voting protocol using threshold schemes. In *Proceedings of 11th Annual Computer Security Applications*, 1995.

[2] J. Benaloh and D. Tuinstra. Receipt-free secret-ballot elections. In *Proceedings of 26th Annual ACM Symposium on Theory of Computing*, 1994.

[3] Michael Cramer, Matthew Franklin, Berry Schoenmakers, and Moti Yung. Multi-authority secret-ballot elections with linear work. Technical Report CS-R9571, CWI, 1991.

[4] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology - ASIACRYPT '92*, 1992.

[5] Ming-Deh A. Huang and Shang-Hua Teng. Secure and verifiable schemes for election and general distributed computing problems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, 1988.

[6] Jinn-Ke Jan and Chih-Chang. A secure electronic voting protocol with IC cards. *Journal of Systems and Software*, 39(2), 1997.

[7] V. Niemi and A. Renvall. Cryptographic protocols and voting. In *Proceedings of Important Results and Trends in Theoretical Computer Science*, 1994.

[8] V. Niemi and A. Renvall. How to prevent buying of votes in computer elections. *Advances in Cryptology - ASIACRYPT '94*, 1994.

[9] H. Petersen and P. Horster. Self-certified keys-concepts and applications. In *Proceedings of Greek Computer Society 1997 Meeting on Communications and Multimedia Security*, 1997.

[10] Michael J. Radwin. An untraceable, universally verifiable voting scheme. `http://www.radwin.org/michael/projects/voting.html`, December 1995.

[11] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, Inc., 1996.